Sorting in general refers to various methods of arranging or ordering things based on criterias (numerical, chronological, alphabetical, heirarchial etc.). In Computer Science, due to obvious reasons, Sorting (of data) is of immense importance and is one of the most extensively researched subjects. It is one of the most fundamental algorithmic problems. So much so that it is also fundmental to many other fundamental algorithmic problems such as search algorithms, merge algorithms etc. It is estimated that around 25% of all CPU cycles are used to sort data. There are many approaches to sorting data and each has its own merits and demerits. This article discusses some of the common sorting algorithms.

**Bubble Sort**  
Bubble Sort is probably one of the oldest, most easiest, straight-forward, inefficient sorting algorithms. It is the algorithm introduced as a sorting routine in most introductory courses on Algorithms. Bubble Sort works by comparing each element of the list with the element next to it and swapping them if required. With each pass, the largest of the list is "bubbled" to the end of the list whereas the smaller values sink to the bottom. It is similar to selection sort although not as straight forward. Instead of "selecting" maximum values, they are bubbled to a part of the list. An implementation in C.

void BubbleSort(int a[], int array\_size)  
{  
 int i, j, temp;  
 for (i = 0; i < (array\_size - 1); ++i)  
 {  
 for (j = 0; j < array\_size - 1 - i; ++j )  
 {  
 if (a[j] > a[j+1])  
 {  
 temp = a[j+1];  
 a[j+1] = a[j];  
 a[j] = temp;  
 }  
 }  
 }  
}

A single, complete "bubble step" is the step in which a maximum element is bubbled to its correct position. This is handled by the inner for loop.

for (j = 0; j < array\_size - 1 - i; ++j )  
{  
 if (a[j] > a[j+1])  
 {  
 temp = a[j+1];  
 a[j+1] = a[j];  
 a[j] = temp;  
 }  
}

Examine the following table. (Note that each pass represents the status of the array after the completion of the inner for loop, except for pass 0, which represents the array as it was passed to the function for sorting)

8 6 10 3 1 2 5 4 } pass 0  
6 8 3 1 2 5 4 10 } pass 1  
6 3 1 2 5 4 8 10 } pass 2  
3 1 2 5 4 6 8 10 } pass 3  
1 2 3 4 5 6 8 10 } pass 4  
1 2 3 4 5 6 8 10 } pass 5  
1 2 3 4 5 6 8 10 } pass 6  
1 2 3 4 5 6 8 10 } pass 7

The above tabulated clearly depicts how each bubble sort works. Note that each pass results in one number being bubbled to the end of the list.

**Selection Sort**  
The idea of Selection Sort is rather simple. It basically determines the minimum (or maximum) of the list and swaps it with the element at the index where its supposed to be. The process is repeated such that the nth minimum (or maximum) element is swapped with the element at the n-1th index of the list. The below is an implementation of the algorithm in C.

void SelectionSort(int a[], int array\_size)  
{  
 int i;  
 for (i = 0; i < array\_size - 1; ++i)  
 {  
 int j, min, temp;  
 min = i;  
 for (j = i+1; j < array\_size; ++j)  
 {  
 if (a[j] < a[min])  
 min = j;  
 }  
  
 temp = a[i];  
 a[i] = a[min];  
 a[min] = temp;  
 }  
}

Consider the following table. (Note that each pass represents the status of the array after the completion of the inner for loop, except for pass 0, which represents the array as it was passed to the function for sorting)

8 6 10 3 1 2 5 4 } pass 0  
1 6 10 3 8 2 5 4 } pass 1  
1 2 10 3 8 6 5 4 } pass 2  
1 2 3 10 8 6 5 4 } pass 3  
1 2 3 4 8 6 5 10 } pass 4  
1 2 3 4 5 6 8 10 } pass 5  
1 2 3 4 5 6 8 10 } pass 6  
1 2 3 4 5 6 8 10 } pass 7

At pass 0, the list is unordered. Following that is pass 1, in which the minimum element 1 is selected and swapped with the element 8, at the lowest index 0. In pass 2, however, only the sublist is considered, excluding the element 1. So element 2, is swapped with element 6, in the 2nd lowest index position. This process continues till the sub list is narrowed down to just one element at the highest index (which is its right position).

**Insertion Sort**  
The Insertion Sort algorithm is a commonly used algorithm. Even if you haven't been a programmer or a student of computer science, you may have used this algorithm. Try recalling how you sort a deck of cards. You start from the begining, traverse through the cards and as you find cards misplaced by precedence you remove them and insert them back into the right position. Eventually what you have is a sorted deck of cards. The same idea is applied in the Insertion Sort algorithm. The following is an implementation in C.

void insertionSort(int a[], int array\_size)  
{  
 int i, j, index;  
 for (i = 1; i < array\_size; ++i)  
 {  
 index = a[i];  
 for (j = i; j > 0 && a[j-1] > index; j--)  
 a[j] = a[j-1];  
  
 a[j] = index;  
 }  
}

Examine the following table. (Note that each pass represents the status of the array after the completion of the inner for loop, except for pass 0, which represents the array as it was passed to the function for sorting)

8 6 10 3 1 2 5 4 } pass 0  
6 8 10 3 1 2 5 4 } pass 1  
6 8 10 3 1 2 5 4 } pass 2  
3 6 8 10 1 2 5 4 } pass 3  
1 3 6 8 10 2 5 4 } pass 4  
1 2 3 6 8 10 5 4 } pass 5  
1 2 3 5 6 8 10 4 } pass 6  
1 2 3 4 5 6 8 10 } pass 7

The pass 0 is only to show the state of the unsorted array before it is given to the loop for sorting. Now try out the deck-of-cards-sorting algorithm with this list and see if it matches with the tabulated data. For example, you start from 8 and the next card you see is 6. Hence you remove 6 from its current position and "insert" it back to the top. That constitued pass 1. Repeat the same process and you'll do the same thing for 3 which is inserted at the top. Observe in pass 5 that 2 is moved from position 5 to position 1 since its < (6,8,10) but > 1. As you carry on till you reach the end of the list you'll find that the list has been sorted. It didn't take a course to tell you how to sort a deck of cards, did it; you prolly figured it out on your own. Amazed at the computer scientist in you ? ;)

**Heap Sort**  
Heap sort algorithm, as the name suggests, is based on the concept of heaps. It begins by constructing a special type of binary tree, called heap, out of the set of data which is to be sorted. Note:

* A Heap by definition is a special type of binary tree in which each node is greater than any of its descendants. It is a complete binary tree.
* A semi-heap is a binary tree in which all the nodes except the root possess the heap property.
* If N be the number of a node, then its left child is 2\*N and the right child 2\*N+1.

The root node of a Heap, by definition, is the maximum of all the elements in the set of data, constituting the binary tree. Hence the sorting process basically consists of extracting the root node and reheaping the remaining set of elements to obtain the next largest element till there are no more elements left to heap. Elemetary implementations usually employ two arrays, one for the heap and the other to store the sorted data. But it is possible to use the same array to heap the unordered list and compile the sorted list. This is usually done by swapping the root of the heap with the end of the array and then excluding that element from any subsequent reheaping.

Significance of a semi-heap - A Semi-Heap as mentioned above is a Heap except that the root does not possess the property of a heap node. This type of a heap is significant in the discussion of Heap Sorting, since after each "Heaping" of the set of data, the root is extracted and replaced by an element from the list. This leaves us with a Semi-Heap. Reheaping a Semi-Heap is particularily easy since all other nodes have already been heaped and only the root node has to be shifted downwards to its right position. The following C function takes care of reheaping a set of data or a part of it.

void downHeap(int a[], int root, int bottom)  
{  
 int maxchild, temp, child;  
 while (root\*2 < bottom)  
 {  
 child = root \* 2 + 1;  
 if (child == bottom)  
 {  
 maxchild = child;  
 }  
 else  
 {  
 if (a[child] > a[child + 1])  
 maxchild = child;  
 else  
 maxchild = child + 1;  
 }  
  
 if (a[root] < a[maxchild])  
 {  
 temp = a[root];  
 a[root] = a[maxchild];  
 a[maxchild] = temp;  
 }  
 else return;  
  
 root = maxchild;  
 }  
}

In the above function, both root and bottom are indices into the array. Note that, theoritically speaking, we generally express the indices of the nodes starting from 1 through size of the array. But in C, we know that array indexing begins at 0; and so the left child is

child = root \* 2 + 1  
/\* so, for eg., if root = 0, child = 1 (not 0) \*/

In the function, what basically happens is that, starting from root each loop performs a check for the heap property of root and does whatever necessary to make it conform to it. If it does already conform to it, the loop breaks and the function returns to caller. Note that the function assumes that the tree constituted by the root and all its descendants is a Semi-Heap.

Now that we have a downheaper, what we need is the actual sorting routine.

void heapsort(int a[], int array\_size)  
{  
 int i;  
 for (i = (array\_size/2 -1); i >= 0; --i)  
 {  
 downHeap(a, i, array\_size-1);  
 }  
  
 for (i = array\_size-1; i >= 0; --i)  
 {  
 int temp;  
 temp = a[i];  
 a[i] = a[0];  
 a[0] = temp;  
 downHeap(a, 0, i-1);  
 }  
}

Note that, before the actual sorting of data takes place, the list is heaped in the for loop starting from the mid element (which is the parent of the right most leaf of the tree) of the list.

for (i = (array\_size/2 -1); i >= 0; --i)  
{  
 downHeap(a, i, array\_size-1);  
}

Following this is the loop which actually performs the extraction of the root and creating the sorted list. Notice the swapping of the ith element with the root followed by a reheaping of the list.

for (i = array\_size-1; i >= 0; --i)  
{  
 int temp;  
 temp = a[i];  
 a[i] = a[0];  
 a[0] = temp;  
 downHeap(a, 0, i-1);  
}

The following are some snapshots of the array during the sorting process. The unodered list -

8 6 10 3 1 2 5 4

After the initial heaping done by the first for loop.

10 6 8 4 1 2 5 3

Second loop which extracts root and reheaps.

8 6 5 4 1 2 3 10 } pass 1  
6 4 5 3 1 2 8 10 } pass 2  
5 4 2 3 1 6 8 10 } pass 3  
4 3 2 1 5 6 8 10 } pass 4  
3 1 2 4 5 6 8 10 } pass 5  
2 1 3 4 5 6 8 10 } pass 6  
1 2 3 4 5 6 8 10 } pass 7  
1 2 3 4 5 6 8 10 } pass 8

Heap sort is one of the preferred sorting algorithms when the number of data items is large. Its efficiency in general is considered to be poorer than quick sort and merge sort.